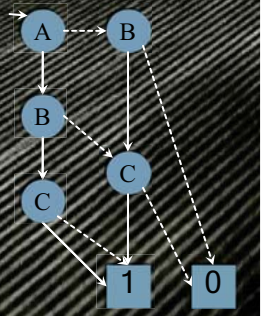
SuffixDD: Sequence BDD に基づく部分文字列索引

北海道大学　伝住　周平

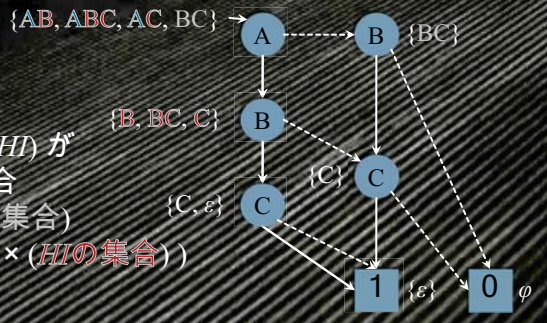
Sequence Binary Decision Diagram

- Sequence BDD [Loekito, Bailey and Pei, 2009]
  - SeqBDD : 文字列集合を表現, 操作するデータ構造
- 根, 内部節点　{AB, ABC, AC, BC} →
  - 二つに分岐
    - *LO*(点線側の子), *HI*(実線側の子)
  - 文字 *C* でラベル付け
  - 三つ組 (*C*, *LO*, *HI*) で表される
- 終端節点
  - 0 : 空集合 φ を表す
  - 1 : 空文字による集合 $\{\varepsilon\}$ を表す



文字列集合の表現

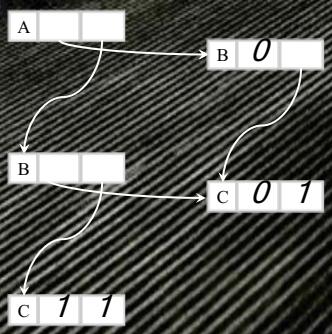
- 根から 1 へのパス 1 本 : 文字列 1 本



- 節点 (*C*, *LO*, *HI*) が表す集合 = (*LO*の集合) ∪ (*C* × (*HI*の集合))

内部表現

- 節点は 3 個の変数 C LO HI をもつ
  - *C* : 文字
  - *LO* : LO側の節点を指す
  - *HI* : HI側の節点を指す
- 0 は 0 を 1 は 1 を指していることを示す
- 64bit環境の場合の例
  - *C* : 8 bit
  - *LO*, *HI* : 28 bit など

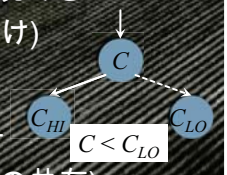


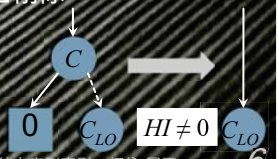
集合演算

- ZDD の演算とほぼ同様に再帰的な集合演算を使用できる
  - 和集合(∪)
  - 積集合(∩)
  - 差集合(\ )
  - *P* {∪, ∩, \} *Q* のいずれも *O*(|*P*||*Q*|) で実行可能
- メモキャッシュの利用
  - 一度解いた計算の結果を記憶する
  - 冗長な計算を省くことができる

制約

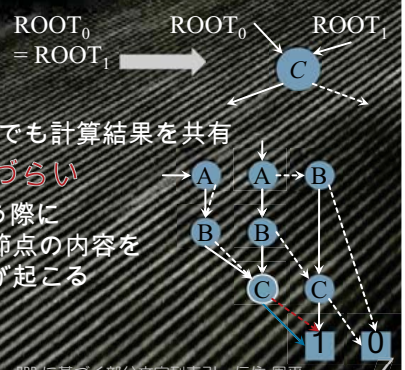
- 二つの制約により, canonical に表現できる
- LO-edge ordered (片側のみ順序付け)
  - (*C*, *LO*, *HI*) の文字 *C* は *LO* の文字より辞書順で小さい
  - 但し, 1 は最大の文字をもつとみなす
- Reduced (冗長な節点の削除, 節点の共有)
  - HI-edge が 0 を指している節点を削除
  - 等価な節点を共有
    - 同一の三つ組 (*C*, *LO*, *HI*) を持つ節点が複数存在してはならない (unique table を用いて実現)





特徴

- 同一メモリ中の節点の共有による長所と短所
- 等価性判定
  - $O(1)$
- 冗長な計算の省略
  - 異なる SeqBDD の間でも計算結果を共有
- 節点の書き換えをしづらい
  - 複数のSeqBDDを扱う際に
    他の根から到達する節点の内容を
    変えてしまうと問題が起こる



内部表現の比較

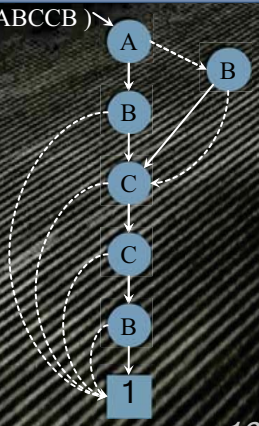
- First child next sibling 形式の辺にラベル付けされたオートマトンと比較
  - 青は非終端状態
  - 赤は終端状態

部分文字列索引

- ある文字列 s が与えられたときに
  その全ての部分文字列を格納するデータ構造
  - 入力：文字列 s
  - 出力：文字列 q が s の部分文字列かどうか
    判定できるデータ構造
- 既存のもの
  - Suffix Try
  - Suffix Tree
  - Directed Acyclic Word Graph (DAWG)

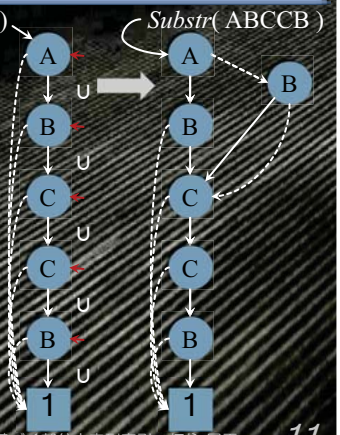
SuffxDD

- Suffix Decision Diagram
  - SeqBDD 上に構築した
    部分文字列索引
  - 入力文字列の全ての
    部分文字列の集合を表現
  - SeqBDD 処理系の
    豊富な演算を使用可能
  - 処理系上で
    アドホックな演算，
    問い合わせが実現可能



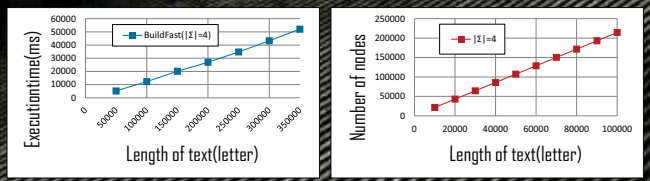
構築時間と節点数

- 構築時間
  - 素朴な方法：$O(n^3)$
    - n は文字列長
  - 効率のよい方法：$O(n^2)$
    - $O(n^2)$ になる例は未発見
    - 期待実行時間：$O(n \log n)$
- 節点数
  - 線形
    - 最小：$n+1$
    - 最大：$3n-4$
  - 辺の数は節点数の高々二倍



実験

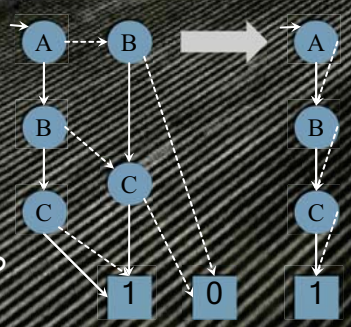
- 入力：一様ランダムな文字列
- 左図　効率良い手法の実行時間
  - $O(n^2)$ ではなく線形に見える



- 右図　SuffixDD の節点数
  - 入力文字列長のおよそ二倍

Multi SuffixDD

- 一本の文字列ではなく文字列集合を SeqBDDとして圧縮したまま入力に与える
- 入力 : SeqBDD S
- 出力 : SeqBDD R
  - $R = \cup_{s \in S} Substr(s)$
- 圧縮されたまま処理し効率を上げる
- 入力のSeqBDDを複数にすることもある



Multi SuffixDD の構築時間と節点数

- 計算時間
  - 高々節点数回の和集合演算を実行
  - 但し, 再帰的に和集合演算をする場合 計算量の解析は難しい
  - $O(\| \{\text{入力文字列集合の異なるsuffix}\} \|)$
  - 正確にはメモキャッシュの効果の調査が必要
- 節点数
  - 入力文字列長の合計に対して線形
  - 個別にSuffixDDを作成するより prefixを共有する分小さくなる